Reference Implementation
Quick Start Guide

Core Services
Reference Implementation
Version 1.2.6

14 December 2005

AFRL/IFSE
JBI Program Office
525 Brooks Road
Rome NY 13441-4505

1. Introduction

Welcome and thanks for your interest in the Air Force Research Laboratory's (AFRL) Information Management Core Services (IMCS) Reference Implementation (RI) Version 1.2.6. This Quick Start Guide should hopefully provide you with a quick understanding of what the RI software does and how you can exploit its capabilities with minimal effort.

1.1 Description

The whole purpose of this document is to provide you – the user, developer, or casual observer – the quickest means possible to take the AFRL IMCS Reference Implementation for a test spin. Once the RI is operating in whatever capacity you choose, you can decide for yourself of its utility as a set of Information Management capabilities and technologies.

If you do not find this document helpful, please let us know. Poorly written or communicated concepts only make both our jobs more difficult.

1.2 Quick Start Guide Layout

If you know what JBI is all about and have used previous RI versions, your best bet is to go directly to the **Quick Start for Old Hats** section. You can always go back at your leisure and read the General FAQ to get the latest scoop on what the AFRL Systems and Information Interoperability Branch is thinking.

If you're brand new to the ways of advanced information management concepts, or want to know what the DoD is doing in response to transformation and network centric operations challenges, you might want to check out the **General FAQ**. If you need a description of, or reason for, the RI's existence, check out the **Reference Implementation (RI) FAQ**. If you want a quick and dirty way to utilize the RI for whatever reason, go directly to **Quick Start for Newbies** where you can run a client locally on your machine that hits an external RI server with a one line script change and a double click.

For anyone who wants to take a little closer look at exploiting the core services from a client developer perspective, we now have a mini-handbook that provides a few rules of thumb for getting the most out of the RI (see **Develop and Run your own Client Applications**). For the first time, though admittedly a risky venture, we provide insight into our test methodology and actual RI performance data in the **Performance Characteristics** section.

Finally, nearly all of the information contained in this document is located in other documents, and is only duplicated here for brevity's sake.

1.3 A Word to Our Early Adopters

A special thanks goes out to the brave souls who have taken delivery of our Release Candidate software, taken it for a test drive, and provided us invaluable technical and usage feedback.

Your comments have highlighted bugs or inefficiencies that we might not have caught prior to final release. Thank you again for your time and effort.

2. General FAQ

The answered questions in this section will hopefully provide curious folks basic information on JBI, information management and how they fit into larger efforts like the Global Information Grid, Network Centric, or Machine to Machine Interoperability concepts.

2.1 What is the Joint Battlespace Infosphere (JBI)?

In one word, "interoperability," or perhaps a couple more, "the technology, legacy systems information engineering and Community of Interest (COI) focus necessary to achieve a level of interoperability beyond what is achieved today using point to point interfaces among traditionally stove-piped systems." Along the road towards achieving flexible interoperability, JBI has a few distinguishing characteristics that make it unique from other approaches:

1. the central requirement to cast data as managed information,
2. loosely exchanging information using a publish and subscribe paradigm whereby edge users are notified of and presented with new information as it becomes available, instead of having to continuously *refresh* for updates,
3. persisting information for subsequent recall, sometimes maintaining references to externally stored *payload* information,
4. collaborating using shared information in a distributed environment,
5. extracting embedded legacy business logic as lightweight decision logic or *fuselets* – a term that is mostly derived from the word *applet* rather than monolithic term *fusion* – to act on behalf of clients within the JBI,
6. dynamically altering the information space as units enter or leave the virtual infospace by identifying their information capabilities and needs via *Force Templates*, and,
7. all of this dynamically administered by an Information Management Staff enforcing Commander's intent and policy in a multi-level secure (MLS) environment.

The JBI is also a concept or vision created by the US Air Force (USAF) Scientific Advisory Board (SAB) in 1999 as it tried for over a decade to get the acquisition community to break down the individual stove-pipes that were created over time and migrate towards an integration substrate that would support interoperability (http://www.rl.af.mil/programs/jbi/documents/TLAP_Final_Volume_1.pdf). In this manner, legacy systems would not have to be replaced but would coexist with newly developed and evolving applications. What distinguished the 1999 study from previous ones are the detailed technical roadmaps the SAB generated and the distinct roles that each identified stakeholder should accomplish to realize the vision.

JBI concepts and vision have been incorporated into the Systems and Information Interoperability (SII) Branch at AFRL within the larger context of advanced information management (IM) capabilities tailored for tactical and operational information domains. Software that implements JBI concepts is being developed within the Branch as Information

Management Core Services (IMCS) - Our "Reference Implementation" has been developed in house and is available for experimentation at no cost.

2.2 What can Information Management Core Services do for me or my program?

If you are a warfighter or C4ISR operator, the core services could provide you a means of getting access to more information than your system currently provides while operating seamlessly within your existing applications. The core services low profile infrastructure and simple client-side access will allow you to exchange managed information with other core service-enabled clients in a transparent manner.

If you are an information manager or perform system administrative tasks, core service capabilities will allow you to dynamically build and manage an information space that can be exploited by users in such a way that enforces Commander's policy as the operational tempo changes.

If you are a legacy systems maintainer or developer that cannot make an external systems interface code change without consulting a Configuration Control Board or modifying a bulky Interface Control Document (ICD), an initial investment to "core service-enable" a legacy system could open up your users to much more information with considerably less development effort. By attacking this so-called *N-squared machine-to-machine (M2M) interface problem* one ICD at a time, this approach will leave small openings in legacy interfaces from which information can be freely exchanged with other legacy systems or core service-enabled clients.

If you are migrating existing systems or developing new applications to operate in a Net-Centric environment, chances are you will have to spend a considerable amount of time identifying your services and the data they will operate on. Much time will also be spent making sure your services are discoverable by external applications. Implicit in this activity is a re-analysis of your business processes and an explicit commitment to embrace new technologies and architectures. Adopting IM concepts now and using existing core service implementations will provide a solid foundation as Network Centric Enterprise Services (NCES) evolve from its current humble beginnings.

If you are a client applications developer looking beyond current Net-Centric Web Service and Services Oriented Architecture (SOA) solutions and moving towards managed information in an information space owned by communities of interest, JBI concepts could represent a next generation capability while still being in synch with current technologies and methodologies.

2.3 What are the GIG, GIG ES and NCES?

The Global Information Grid (GIG) provides the foundation for net-centric operations by globally interconnecting the building blocks of information capabilities, including: the physical communication links, hardware, software, data, personnel and processes. The GIG Enterprise Services (ES) is an umbrella term to describe the information services that reside on the GIG. The Network Centric Enterprise Services (NCES) is a DISA-sponsored program that seeks to provide a generic set of core services that will be available to all DoD edge users allowing them

to "pull mission-tailored information intelligently from anywhere within the network environment [1]."

The initial NCES core services are being developed using the latest industry standards, such as extensible markup language (XML) and web services, and will provide functionality as services to GIG edge-user applications. This architectural approach and its use of standards and web technologies are referred to as a Service-Oriented Architecture (SOA).

2.4 What do the Information Management Core Services have to do with NCES?

The Information Management Core Services will be built to run on top of, or along side, NCES core services and when deployed, will establish an interoperable *information space* that aggregates, integrates, fuses, and intelligently disseminates relevant information to support effective warfighter business processes. This virtual information space provides individual users with information tailored to their specific functional responsibilities and provides a highly tailored repository of, or access to, information that is designed to support a specific COI, geographic area or mission. This information space also serves as a clearinghouse and a workspace for anyone contributing to the accomplishment of the operation—for example, weather, intelligence, logistics, or other support personnel [2].

In combination with NCES core services, future Information Management Core Services will be utilized to dynamically create an information space and manage edge user interaction within a COI. An example of this, from a GIG ES perspective, would allow edge users to discover relevant COIs (*using NCES*), gain access to the information space (*IM access controls using NCES user profiles and underlying security mechanisms*), and exchange or manipulate relevant information (*using IM core services*). In this manner, an IM-enabled COI would have the full suite of core service information management capabilities, but would still be based on and have full access to NCES services. Together, NCES and IM core services will be utilized to develop end user applications targeted at mission-specific COIs to facilitate a secure, controlled information exchange and decision support environment.

Within the GIG infrastructure, multiple IM-enabled COIs and other edge user applications will be established to satisfy specific end user needs. IM-enabled COIs will be capable of sharing their information among each other as well as other edge users operating on the GIG. The net result is that each user will have capabilities above and beyond what any individual service capability provides.

The current Reference Implementation is not NCES compliant, although several prototypes have been built on top of the limited set of initial NCES Discovery services. The previous discussion is conceptual in nature; however, V1.2.6 has a Web Services interface to its publish and subscribe engine which is a first step on the path to NCES compliance. Future versions will be more tightly coupled to the evolving NCES services.

2.5 What is the N^2 Machine-to-Machine (MTM) Interface Problem?

In a 2003 Summer Study on Technology for MTM Intelligence Surveillance, Reconnaissance (ISR), the USAF Scientific Advisory Board presented four options for dealing with interfacing

disparate systems, architectures or domains into a cohesive, integrated architecture. The de facto standard in how we interface existing systems is called the pair-wise integration of systems whereby the cost of integrating N systems scales as N^2, but is likely less than that since you don't have to integrate all the pairs (see Figure 1). Other options include integrating N systems into a Grand Unified Architecture or by grouping a subset of N systems into P domains and integrating the domains, but evolution to these architectures is not feasible due to costly retrofit requirements or "big bang" approaches, respectively. The option the SAB settled on revolves around a preferred architecture as a central hub in which you integrate the various domains to achieve horizontal integration with cost N+P+1, called a *Federated Architecture*. To tie this all back to the Information Management, a preferred architecture would revolve around NCES, while the IM Core Services would be best suited to integrate within the domains – each configured as their own Community of Interest.

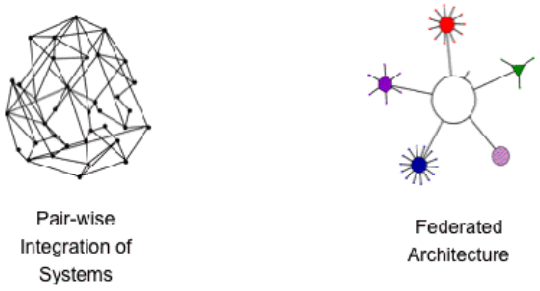

Figure 1: Architecture Options for Interfacing Disparate Systems

3. Reference Implementation (RI) FAQ

The questions and answers in this section relate to more mundane issues like AFRL's interpretation of grander concepts and visions.

3.1 What is the Information Management Core Services Reference Implementation?

An AFRL-owned and -developed prototype software capability that implements a subset of the JBI concepts listed above. The AFRL in-house team designs, develops and manages the configuration of the RI using the USAF Scientific Advisory Board's original JBI vision as a guidepost. The RI has two roles: one, as a research vessel for small development groups to develop plug-ins that test out theories, algorithms and hunches; or as a proof-of-concept implementation whereby developers can build clients that interact with a real set of core services using a standard common API (CAPI). The latter role is most suitable for technical experimentation, but could also be used as the first step in technology *transition* if packaged with client development or as a *transfer* of ideas or concepts

Each version of the RI has unique capabilities and performance characteristics. An RI Version Description Document (VDD) defines previous version capabilities and highlights future plans. As a general rule of thumb, the RI strives to add functional capabilities at the expense of performance, although new features are carefully analyzed to ensure the core performs at a reasonable level. Any functional-performance tradeoffs will be documented in the Quick Start Guide so that potential users are informed before spending time and effort tinkering on their own.

Other commercial implementations and related information management technologies exist, such as Boeing's X-Infosphere project and the AFRL-sponsored Mercury initiative. These two implementations also adhere to the CAPI specification so clients can transparently use either underlying core. Currently the cores themselves are not interoperable, but the latest in house work is geared toward crafting specifications with a "system of systems" view to close that gap as well.

3.2 What is the CAPI?

The Common Applications Programming Interface (CAPI) provides a consistent interface to anyone wishing to develop client applications or implement their own subset of IM core services. As the common interface to any core services implementation, it is envisioned that all communication between the core and clients is through the Common API. This allows for the development and deployment of several different core service implementations in many programming languages without impact to clients and the information objects they exchange. The CAPI specification is managed by a group of like-minded scientists and engineers that comprise the www.infospherics.org community.

3.3 Why do I care about the RI?

You don't have to, but it's nice to know you can go somewhere to find real software that begins to implement some of the JBI concepts. Plus, it's free and who passes up free stuff?

Also, much effort has been expended to build the RI with open source components so that it is available at no cost. We've even gone so far as to purchase a redistribution license for certain 3rd party components so we could remain a zero cost option for anyone wishing to use the default configuration.

3.4 Why go through the trouble of building the RI?

We're in a much better position to evaluate potential COTS products or other technical solutions for real application, having gone through much of the pain and learning curve ourselves. We've become contributors – in both concepts and software – to other RI developers so that true interoperability might be realized someday. We have contributed back to the open source community from which we've borrowed designs and implementations. Plus, we have the satisfaction of actually having built something beyond PowerPoint presentations.

3.5 What are the features of the latest version?

The biggest change is that CAPI 1.2 backward compatibility has been removed from 1.2.6. By removing CAPI 1.2 backward compatibility we improved the performance and maintainability of the RI significantly. Also removed, is the ability to migrate from previous RI repositories. Yep sad but true, you will not find a migration utility with the JBI V1.2.6 RI.

The IOR/MSR implementations have been optimized to allow more efficient removal of types (schemas). In V1.2.5 of the RI the time it took to delete a schema from the MSR grew linearly with the size of that types IOR. This in effect made the IMS Tools page appear to "lock up" If a

user tried to delete a schema via the IMS tools. In extreme cases some users would click the delete button multiple times, hoping to speed the process along, instead it made the time to delete grow exponentially!! This of course would result in out of memory errors, database corruptions, performance degradation… in short problems, problems, problems. It's worth nothing that although we have dramatically improved the time it takes to delete schemas with large numbers of IOR entries, deleting InfoObjects from the IOR from the IOR page in the IMS Tools can still take quite a bit of time with a large IOR. As a general warning we recommend only hitting the delete button once and giving the RI time to do its work. In other words, if you've got 6 million records and decide to delete only 1 million of them it may also be a good idea to take some of the annual leave you've stored up and just come back the next day. This strategy works particularly well if the current day is Friday and the "next day" will be Monday.

Version 1.2.6 provides a toolset for Information Management Staff (IMS) personnel to build, manage and monitor an information space that is exploitable by external client applications. The IMS toolset allows administrators to construct managed information objects, create client and group accounts, establish client privileges as an extension of Commander's guidance, and administer repositories that store metadata schema, archived information objects, and security information. Once configured, an Information Space is then used to support client applications wishing to exchange managed information in a consistent manner via publish, subscribe, archive and query functions subject to dynamic privileges invocation/revocation. The following table describes these capabilities in more detail.

Major Feature	**Detailed Description**
Common API	• CAPI V1.5 (see www.infospherics.org) that defines a consistent interface to the core for client applications – the Reference Implementation is the embodiment of this independent CAPI specification • CAPI V1.5 is not backward compatible with CAPI V1.2. • CAPI V1.5 includes syntactic and semantic changes: setting attribute and type descriptors and changes to query sequence interaction. There are still some unimplemented methods, so consult the Javadocs for the latest status. • Java and Web Services Description Language (WSDL) mappings to the CAPI (*C#-WSDL mapping coming soon to infospherics*) discoverable through UDDI registry
Core Services Connection	• Clients can simultaneously connect to multiple RI implementations that provide a compatible client side connection plugin. • UDDI registry provides dynamic lookup of platforms and individual services for Java and Web Service (WS) clients. WS clients automatically load endpoints/access points, while Java clients dynamically load local classes prior to interaction • Discovery Services for dynamic client location of platforms and their CAPI services. Clustered configuration will use these services to register, initialize and synchronize platform members. • Pluggable connection manager allows developers to implement their own core services with minimal impact to client (i.e., interaction would be the same, but clients would need different libraries)

Major Feature	Detailed Description
Information Object (IO)	• Info Objects composed of type, metadata (characterizes the payload) and payload • Type descriptors used to describe parent-child relationships among multiple info objects • Base Object is transparent and implied root parent to all info objects. Base Object metadata tags are "hidden," allowing any XML schema be used, "as-is" without forcing users to add extraneous XML elements. Clients have the option to view these extended tags/elements if they wish • User-defined Metadata based on W3C 2001 XML Schema XSD Simple Types (All 44 types supported) • Payloads can conform to most MIME Types, including structured data such as XML documents or unstructured such as video or images
Information Dissemination	• Publish and Subscribe paradigm using pluggable proliferator-receptor chain (uses Java Message Service [JMS] default); but has provisions for other experimental plug-ins • Brokering using pluggable predicate evaluation (DOM4J provides full XPath processing supported as default) • Query Brokering using full XPath 1.0 predicates. Actual queries are processed as XQuery expressions. • Client-side XQuery expressions are not supported.
Persistence	• Archived Info Objects with separate metadata and payload storage using Native XML and name/value pair databases, respectively. Collectively known as the Information Object Repository (IOR). • Metadata Schema Repository (MSR) for defining information object schemas. • Client and Group Accounts and Privileges data store • Default persistence using Sleepy Cat's Berkeley XML DB, with unsupported plug-in for Raining Data's TigerLogic data store.
Security	• Encrypted Client-Server interaction and password protection • Fine-grained access controls down to the information object type and version level, based on the type of operation performed on the information • Java objects used to store subject (principal), group and role information • Employs initial XACML policy representation and enforcement during run-time access – supports "negative" privileges as highest precedent • Client and group-level authorization
Instrumentation	• Server-side Instrumentation API (ICAPI) utilized by core services to report client interaction data (connections, sequences, IPs, information object exchange rates, etc.) • Configurable with on/off and report frequency settings to minimize impact to the core services during run-time • Visualization client that depicts client and platform nodes as a connected graph. Includes statistical information on core service resource utilization.
Information Management Staff (IMS) Tools	• Web services allow management of MSR, IOR and Security Repository Service (SRS). The MSR supports the drop-in of any user-defined schema; IOR management includes deletion functions; and SRS allows graphical manipulation of positive and negative privilege and group assignment • Java Management Extension (JMX) Console used to tweak performance variables and manage internal application server components

Major Feature	Detailed Description
	• Apache Axis integrated as WSDL Viewer for Web Service clients • Instrumentation services and visualization client for monitoring client usage of core services • Google-like feature allows browser clients to submit keyword value searches against the IOR (unsupported)
Architectural Considerations	• Built on top of Java 2 Enterprise Edition (J2EE), and specifically, the open source JBoss version 4.0.2, using WS4EE web services and service oriented architecture (SOA) front-end. This configuration supports UDDI services and implementations. • JBossCache provides a persistent, distributed cache for maintenance of internal state and instrumentation data. • Server-side components can be distributed or clustered for better performance under certain conditions. By default, the application server is configured as a cluster of one. • Application server auto-configures itself the first time it is launched – users are taken through a series of simple questions that are used for subsequent startups • MySQL Version 4.1.12 as the underlying default Relational Database Management System (RDBMS) • Berkeley DB XML 2.2.13 native XML database from Sleeepycat Software. • TigerLogic XML database plug-in available (unsupported) • Core services run on all Windows platforms, Solaris 8 and 9, all Linux variants and Mac OS X.
Sample Clients	• Example Applications that discover the V1.2.6 services dynamically • Extensible Subscription Viewer for quick subscriber capability • Schema Builder client conforms to V1.5 Base Info Object format • Training Client and tutorial • Test Applications included to provide the adventurous with a means to gauge performance of their server configuration • Utility Application for quick pub/sub/query

3.6 Where can I get a copy of the RI?

Fill out the form at https://cmweb.deepthought.rl.af.mil/pls/new_user/new_request.cmdb. This web site is managed by AFRL's own Configuration Management (CM) Group.

4. Hardware Options

To cut down on at least a thousand variables when choosing a hardware platform to run the RI software, we've boiled the choices down to five options: a good configuration, a better one, a recommended one, an even better configuration, and finally an enterprise configuration. In true engineer fashion and because this document is intended to be a *quick start*, we'll only actually talk about two of these configurations. One that simplifies installation for newbies, and another that allows experienced users to quickly use the new features. So, if you're one of the latter people, feel free to skip ahead to your section and you'll be referred back here as appropriate.

Configuration	*Client Hardware*	*Server Hardware*	*Separate Repository?*	*# of Client Apps*
4.1 Good Configuration	Same machine as the server (*running both client and server*)	High Performance Laptop (> 2Ghz, > 768Mb RAM)	No	1
4.2 Better Configuration	Desktop ≥2.4GHz 1MB L2 Cache MS Windows 2000 Pro	High Performance Desktop (> 2.4 Ghz, Dual P4-Xeon 512kb Cache, 2Gb RAM)	No	2 - 10
4.3 Recommended Configuration	*Same as Better Configuration*	Entry level Server Class Machine, (Dual P4-Xeon >1Mb Cache, 4Gb)	No	11 - 50
4.4 Even Better Configuration	*Same as Better Configuration*	Small Server Class Machine, 2 Processor RISC-based High Performance Blade Server	No	51 - 250
4.5 Small Enterprise Configuration	>1 x Desktop >3GHz ≥1 GB RAM	Medium Class Server, 4 processors or greater RISC-based, High Performance Blade Server	Yes	251 - ~

If you're still reading, note that a good configuration is simply a very capable laptop or desktop equivalent. This will allow multiple clients interacting with a single core to reside on a single, configurable machine. With minimal effort the good configuration will allow a quick absorption of IM concepts with acceptable, demonstrable capability. From better to even better configurations, we take the approach of having the same capable client machine, which is realistically a Windows machine, and point it at a server residing on a Windows machine and two Unix variants, Linux (*must build your own download and build your own Berkeley v2.2.13 http://www.sleepycat.com/ the source is also available on the JBI V1.2.6 RI distribution CD*) and Solaris. The single server concept allows the core service components to reside on a single machine, while the enterprise configuration introduces more client machines and eludes to speading the core components across machines(like the various repositories) in order to increase performance.

5. Quick Start for Newbies

To the right, you'll see the distribution CD structure. For this section, you should only need to know that the client directory contains the client installation wizard, and the server directory contains the server executable code.

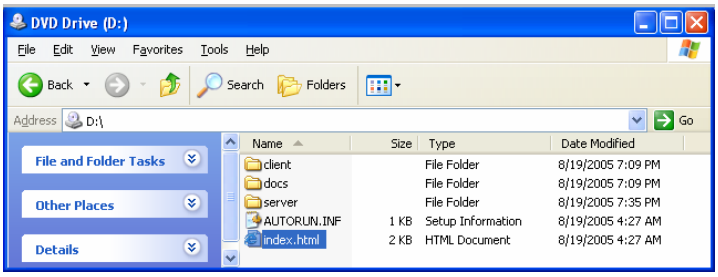

Figure 3: Distribution CD Directory Structure

5.1 Choosing your Hardware Option

We'll make the choice for you: get yourself a capable laptop described above as our **Good Configuration** and follow along.

5.2 Software Configuration Options

For a newbie, you have two options: one, whether you want to publish and subscribe as quickly as possible using an existing server; or two, whether you want to quickly establish your own server and use the sample applications provided on the CD. All other decisions are made for you at this time.

5.2.1 Running Sample Client Applications against an Existing Server

Reference the following sections in order of the RI Tutorial and Sample Applications Users Guide for details on using the sample "out-of-the-box" applications: 3, 4, and 6. You will need to choose either the on campus or off campus servers.

5.2.2 Installing and Configuring your own Server

Reference the RI Core Services Configuration and Installation Guide Sections 3-6, keeping in mind the following exceptions:

1. In Sections 3.2 and 3.3, only Windows and Solaris 8/9 are supported.
2. MySQL 4.1.12 is recommended. More specifically avoid 4.1.8, or 4.0.x as they will not work with the RI.

5.2.3 Running Sample Client Applications using your own Server

Step 1 – Install and Configure your own Server

Execute the steps in Section 5.2.2 of this document, exactly as they appear.

Step 2 – Login as an Information Management Staff (IMS) administrator

Launch a browser and visit the IMS Tools at https://localhost:8443/RepositoryTools/. When prompted, use username **jbiAdmin** and password **moniker**. You will then be taken to the main IMS page where you can manipulate the Metadata Schema Repository (MSR), Information Object Repository (IOR) or Security data store that contains client accounts, groups and privileges. Make sure to change the default password when you get a chance.

Step 3 – Add Schemas into the MDR

You will need to install the Client Side Applications before following these examples. See section 3 of the RI Tutorial and Sample Applications Users Guides in order to install the Client Side Applications.

Click the "MSR" link, and then click the "Add Schema" link. Fill out the fields based on the first row of the table below, using the "Browse" button to find the specified file within the JBI Client-side installation, and click the "Submit" button. Repeat until all table rows have been input.

IO Type Name	**IO Ver. #**	**Descrip-tion**	**Schema Location Root path:** *jbi\afrl\v1.2.5\schemas\mil\af/rl\jbi\training*
`mil.af.rl.jbi.training.ato`	1.0	Training Object	*`ato\mil.af.rl.jbi.training.ato.xsd`*
`mil.af.rl.jbi.training.basic`	1.0	Training Object	*`basic\mil.af.rl.jbi.training.basic.xsd`*
`mil.af.rl.jbi.training.xmlxpath`	1.0	Training Object	*`xmlxpath\mil.af.rl.jbi.training.xmlxpath.xsd`*

For more detailed discussion on how to use the IMS Tools, see the RI Information Management Staff Administrators Guide.

Step 4 – Add a Default User

To successfully launch the Sample Applications, you must add the default user **jbiuser** with password **test**. Give the user * privileges over the types. See Section 4.2.1.3 (Add User) of the *RI Information Management Staff Administrators Guide V1.2.5* for more detail.

Step 5 – Launch a Sample Application

Reference Section 6.1 of the RI Tutorial and Sample Applications Users Guide for details on using the sample "out-of-the-box" applications. You'll only need to use the IP address of the machine you launched the core services on.

5.2.3.1 Accessing the Tutorial

The JBI Client-side tutorial can be found within the RI Tutorial and Sample Applications Users Guide.

6. Quick Start for Old Hats

So, you've already received an official copy of V1.1, V1.2, V1.2.0.1, V1.2.5, V1.2.5.01 or a bootleg copy of V1.1.1_TheAtticMix and you'd like to quickly utilize the new capabilities. This section will point the way. If you're interested in the deltas, we highlight the changes in the next section so you don't have to guess or learn the hard way like you might have had in upgrading from older versions. For client migration, we identify the minimal steps you need to take to port your clients.

6.1 Changes from Previous Version

Major Feature	Detailed Description
Common API	• CAPI V1.5 compliant (includes improvements to syntax and semantics, like: implied Base Object as root, type descriptors for parent-child relationships and types/versions, query sequence changes, and massive implementation class refactoring); still some unimplemented CAPI methods dealing with attribute descriptors • Java and web service clients supported using dynamic discovery of services • More descriptive Javadocs available
Core Services Connection	• UDDI for dynamic registration and lookup of RI platforms and their services • Discovery Service for dynamic client discovery of platform locations and their CAPI services. In this implementation CAPI services do not have to be running on the same server as the "platform" - uses platform connection descriptor as implementation • Web Services clients use dynamic discovery and subsequent endpoint (access point) loading to interact with platform • Java clients use *client.properties* file to find UDDI registry
Information Object (IO)	• Base Info Object will have minor metadata changes (fields reordered, coercion dropped, string-to-date, etc); however, specifying the base info object in user-defined schemas is no longer necessary • Publisher-provided metadata elements transparently wrapped in system-provided tags • Repeating elements and XML attributes are now supported • Supports drop in of any user-defined schema (e.g., DISA XML registry) • Externally referenced and included Namespaces for user-defined metadata are also supported
Information Dissemination	• Architecture modifications to enable any participating partitions to match publishers and subscribers for more efficient load balancing • Improvements to subscription matching allowing XPath nodeset and predicate syntax; XPath encapsulated in XQuery expressions • Full XPath 1.0 support on both subscription and query. Even though queries are turned into XQuery expressions, XPath 2.0 is not supported due to internal parser grammar • XQuery support is not provided.
Persistence	• Native XML database for storage of metadata instances in the IOR • IOR payloads now stored in name/value pair database; RDBMS remnants only for persisting cache (including Java objects), JUDDI registry data,

Major Feature	Detailed Description
	session values and MSR information; in-memory cache for query processing • Unsupported TigerLogic XML database is available as an evaluation plug-in to test out persistence. The TigerLogic/MySQL combo will replace the Berkley XML DB/Berkeley DB pair for TigerLogic evaluation
Security	• New Security Infrastructure using XACML policy representation and Security Repository Services (Java objects) instead of security triplets; Includes support for groups, content-based access control ANDed to user-provided predicates, and negative privileges • Policy can be modified by adding syntactically correct XACML files in application server deployment directory • Security privileges for accessing Instrumentation client (authentication only - no authorization) • Only positive and negative privileges are enforced through XACML, all others use Java attributes – no more ugly pound-sign delimited roles
Instrumentation	• Functional Requirements – answers the who, what, when, where, how many with historical references, including clustered core partitions • Interface Requirements – includes API for core service usage; and client usage through more formal API • Run-Time Requirements – minimal impact to core performance; ability to turn instrumentation on and off • Client Requirements - Java application capable of visualizing clients, connections, sequences, core components and others in a node-graph representation • Turning the instrumentation services on and off is configured through the JMX console; settings should remain between application server startups
Information Management Staff (IMS) Tools	• Group assignment, additional security predicate specification as content-based access control, and negative privileges assigned through new IMS web page GUI development; setParent capability supported • Separate Java application that illustrates instrumentation service capabilities • Google-like search capability using standard browsers (unsupported).
Architectural Considerations	• Built on top of Java 2 Enterprise Edition (J2EE), and specifically, the open source JBoss version 4.0.2, using WS4EE web services and service oriented architecture (SOA) front-end. This configuration supports UDDI services and implementations. • Server-side components can be distributed or clustered for better performance under certain conditions. By default, the application server is configured as a cluster of one. • Application server auto-configures itself the first time it is launched – users are taken through a series of simple questions that are used for subsequent startups • MySQL Version 4.1.12 as the underlying default Relational Database Management System (RDBMS) • Berkeley DB XML 2.2.13 native XML database from Sleeepycat Software. • TigerLogic XML database plug-in available (unsupported)
Performance	• Clustering default set to "a cluster of one" although multi-member clustering nodes supported • Full performance numbers and/or optimal configuration settings may not be available

Major Feature	Detailed Description
Sample Clients	• Fully ported Sample Applications, JVCU, TestApps and Training Client to new CAPI • Schema Builder uses latest Base Info Object schema • Includes How-To guide to port V1.2 clients to V1.2.6

6.2 Considering a New Hardware Option?

Well, you might not really need to upgrade. But for quick start purposes, we recommend the Better Hardware Configuration and will stick to it as we describe how to move to the newest version.

6.3 Porting your Clients

If you updated your code to work using the 1.5 CAPI api. Then first breath a huge sigh of relief and second know that life with V1.2.6 will be pretty good after you swap some jars.

If you relied on the CAPI 1.2 backward compatibility provided in the 1.2.5 RI note well that your apps WILL be broken when you try to use them with V1.2.6 RI.

Reference the RI Client Migration Guide for a step-by-step process on transitioning your V1.2.0.1 or V1.2.5 clients to V1.2.6.

6.3.1 Configuration

Reference the RI Client Migration Guide Section 5 and 4.1 – 4.2 to review the changes from the previous version and changes to the client.properties file. Note that from V1.2.5 To V1.2.6 there have been no changes to the format or content of the client.properties file.

6.3.2 Run-Time Libraries

Reference the RI Client Migration Guide Section 3.1.

7. Develop and Run your own Client Applications

There currently is no guide or handbook for client developers, although we are working on it. To fill that void, you can check periodically at www.infospherics.org or use this quick reference to alleviate any unnecessary JBI growing pains.

7.1 Is JBI and Information Management the Right Paradigm?

If you want to explore the publish and subscribe paradigm among loosely coupled clients above and beyond what standard message-oriented middleware (MOM) products provide, the JBI RI might meet your needs. The JBI premise is based on the management and dissemination of managed information objects (MIO), characterized by type, metadata and payload, where brokering among the objects is conducted using XPath processing on its metadata. Key to the exchange of information is also the ability to manage it from an administrator's perspective

where client privileges can be invoked or revoked dynamically at the information object level based on the operation performed on the object.

If you want to create an *integration substrate* that establishes some level of interoperability between one or more systems, you'll need to take a hard look at the data that is currently exchanged between the systems and *engineer* information objects that can be exchanged using publish, subscribe and query. While we briefly discuss rules of thumb for information engineering in the next sub-sections, our focus over the next few releases will be to provide you with more definitive ideas on how to accomplish this. Suffice it to say, that wherever there exists an ICD between two systems, you could replace the interfaces with two clients on either side with JBI in the middle.

If you want to do real-time processing and dissemination of raw data such as radar feeds you should look elsewhere. The JBI works best on *information* not *data*. If one defines information as data imbued with context and meaning, then the MIO-type supplies the meaning of the information and the metadata captures its context. The metadata and type and payload together are managed and disseminated by JBI. Often, low-level data being generated at high rates is not associated individually with meaning and context. In short, the data elements are not intended for individual processing or dissemination - rather they are treated as a set (or stream) and the assumption is made that anyone listening to the stream should implicitly understand its meaning and context. While these tightly-coupled systems are indisputably valuable in many mission-critical applications, they do not tend to be very flexible or adaptable. Information management systems, such as JBI, are built to provide flexibility, but not every system needs this capability and some systems cannot afford the performance implications of a managed information environment.

7.2 Information Engineering

A GAO special publication (www.gao.gov/special.pubs/bprag/bprgloss.htm) defines Information Engineering as an approach to planning, analyzing, designing, and developing an information system with an enterprise wide perspective and an emphasis on data and architectures. This will suffice for JBI purposes except that the architecture is the RI and engineered data will be contained within information object schema and instances.

7.2.1 A Word on Managed Information Objects

The managed information object (MIO) is the *unit of measure* within the JBI. Its structure is defined and *registered* in the Metadata repository by IMS staff or other clients before instances are published or received via subscription or query. The MIO comprises two parts: the metadata and the payload. The payload is the information that is being distributed and the metadata is the "information about the information" that describes the payload and allows for the matching of the information object to existing subscriptions and future queries.

Information objects are immutable. If the information that is carried by the object changes, during processing by a client or a fuselet, a new information object is created. Information objects are not objects in the object oriented programming (OOP) sense - methods and data are not bound together into a single conceptual unit. Information object instances are comprised of

their attributed-value pairs within the metadata and are decoupled from any actions (methods) that act upon them.

7.2.2 Managed Information Object Types

Managed Information Objects must have a unique name or *type*. A set of information object types can be organized into a hierarchy to better manage their related metadata. An implicit hierarchy is in place even if a JBI only manages a single MIO because every MIO inherits metadata from the base MIO type (discussed in the next sub-section). The goals of the object type hierarchy include:

- Organizing the set of types to facilitate human and machine searching of the related metadata elements
- Simplifying the expression of policies over the type hierarchy, and
- Providing a mechanism for obtaining all child subtypes by subscribing to its parent type

MIO types can be arranged in a hierarchy as parent- child relationships, whereby a child schema extends its parent schema to include inherited metadata. In this manner, multiple siblings can share common elements that can then be used for subscription brokering at the child level.

MIO types can also be structured in a tree abstraction called a Community of Interest (see the RI Information Management Staff Administrators Guide for a more thorough discussion). Implementation-wise, it still is a hierarchy much like the parent-child relationship except that no schemas are required to establish a node in the COI hierarchy – it's just a grouping much like directories are to files at the operating system level. Computer users don't store all of their files in one giant directory, nor do they use nondescript file names like *myfile1*; they take the time to build a meaningful directory structure (*COI*) in which to place their well-named (*typed*) documents. COI structure is usually formed to classify MIOs functionally if they're critical to a process (e.g., by tactical operation) or organizationally if agencies require specific management oversight on their information.

7.2.3 Metadata

Metadata is currently represented as XML and defined in an XML Schema. The single metadata schema has two parts: the base elements that every JBI information object should have and user-defined elements that make the object unique.

7.2.3.1 Base Information Object

The elements that comprise the base object schema are listed in Appendix A.

7.2.3.2 User-Defined Metadata Elements

When specifying your own user-defined metadata elements, you no longer need to include the base information object or any other core service-specific XML elements. These are now transparent and support basic "drop in" of schemas from such places as DISA's XML Registry.

When specifying your user-defined metadata, you can create elements or sub-elements as complex types from a single namespace, namely http://www.w3.org/2001/XMLSchema (see Appendix B for supported simple types). References to other namespaces will be ignored. Repeating metadata elements are now supported as are XML attributes contained within an XML element's tag. Still unsupported at this time is including external schemas in your main schema definition by way of using **xsd**:**include.**

For user-defined MIO types that fit into a hierarchy, there is no mechanism to reference parent MIO types at the child level or vica versa. The parent MIO's elements must be explicitly included in the metadata schema and published instances.

7.2.4 Payload

Payloads can contain either structured or unstructured information. Links or URLs can be stored in the payload as a reference to the actual content, but it is up to client applications to retrieve the content on their own. There is currently no native accommodation for such out-of-band payload retrieval the RI.

7.2.4.1 Structured Information

If your information objects can be represented as a single XML document, you basically would have structured payload after partitioning the elements into metadata and payload. Deciding on whether an element should be metadata or payload is akin to deciding whether you should use an element or attribute in an XML document – you get a different perspective from each person you ask. The only rule we can offer is to stuff everything into the payload and then promote elements that clients would likely subscribe to or query against. If it's not in the metadata, it can't be brokered. From past experience, data elements that would normally be primary keys, foreign keys or have unique values in databases should be metadata. The same holds true for geospatial- or time-related elements - particularly for military applications that use maps. Finally, any *types* that often appear as enumerated lists, such as mission, vehicle, country code, or target types should be user-defined metadata.

7.2.4.2 Unstructured Information

In theory, just about any MIME type can become a payload, provided it can be properly converted into a Java serialized object. So far, we have not found any restrictions on this – client applications have disseminated and processed all types of image files, video, MS Office files, and others.

7.3 CAPI Familiarization (Interacting with the Core)

Point your browser to infospherics.org and download the Java language mapping to the CAPI specification in javadocs format. Or, once you have the V1.2.6 core running, you can obtain the Web Services Description Language (WSDL) from the Axis Console on the IMS pages. In addition, you should take a look at the source code for the example applications in the client directory on the distribution CD and compare against the CAPI.

7.4 Developing your application

Before developing a JBI application, you'll have to ask yourself a few basic questions the previous sub-sections were trying to answer. First, is JBI the right solution for your purposes? If so, have you looked at what constitutes information within your application in order to recast it in the form of information objects? Have you narrowed your development environment down to Java or another programming language that can use the RI WSDL definitions? Do you have a copy of the CAPI Javadocs?

After reviewing the CAPI, and example application source code, you'll notice that a basic blueprint emerges for most clients as they interact with the core.

CAPI Operation	Description
1. Edit the `client.properties` to point to core	Not a CAPI operation, but configuration step; RI clients still must know where the core is located by using IP or DNS
2. Connect to core with specified credentials	Interaction encrypted via SSL; Authentication occurs under the covers – unauthenticated clients will be bounced
3. Establish Pub/Sub/Query Sequences	Access Control based on privileges (security triple) occurs at sequence initialization; happens behind the scenes
4. Set predicates/attributes and activate sequence	Can pause and resume sequences - signifies sequence state change (configurable access controls working transparently)
5. For publications, create info objects and publish	Info objects created within the context of the sequence, not outside context
6. For subscriptions, will be notified of receipt of info object in client's address space or polling may be used	May implement call back method to handle notification and receipt of info object
7. For query, must specify result set size before activating, then loop through until consumed	Result sets sorted in descending order of publication time – this allows retrieval of last archived info object with result set size of one
8. Close sequence	Self explanatory
9. Disconnect	This is always recommended so the core can clean up and reallocate resources

For Web Services development, check back with infospherics.org over the Fall of 2005 for alpha versions of C# mappings to the CAPI.

7.4.1 CAPI Client Command Line Arguments

See section 4.2 of the RI Tutorial and Sample Applications Users Guides to learn more about the CAPI client command line arguments.

7.5 Running your application

Though we cannot tell you what the best way to run the application you have developed is, we can give you hints on what you can expect. First, you must include key libraries in your class

path if it uses the RI for information management and dissemination. If you are using another 3rd party application that loads the class path (i.e. Eclipse) make sure the application loads the class paths properly, or that the application's properties are set appropriately in order to enable proper class loading.

Now that JBI V1.2.6 RI can be clustered you might want to pay attention to the binding IP addresses when configuring your server. If you have more than one network card you may be binding the IP address your clients utilize to a specific value (if not, the first IP found on your server will be used as the default). It is important that the IP address set as the "binding address" is consistent with the value set in your using client's client.properties file. If not, your clients will not be able to communicate with the server.

The RI's CAPI implementation has been tested extensively using multithreaded clients. As a general rule be careful with clients that use multiple threads and share Connections or Sequences. Keep in mind that while the CAPI itself is thread safe that does not mean your usage of it as a client is guaranteed to be as well. For example, if you instantiate a global static instance of a Connection that is then used by multiple threads, note well that you could unintentionally change or invalidate the state of the connection in one thread rendering it and it's sequences useless for your other threads.

When creating schemas try to choose the best XSD type for each element. The schemas are now used to index the metadata according to their XML element types. For example, simply choosing xsd:any when the type could more accurately be represented as xsd:decimal will cause inaccurate indexing of the information stored within the Berkeley XMLDB and would, ultimately, result in worse query performance. This particular case is very bad as for xsd:any the RI does no indexing at all!

8. Performance Characteristics

This version has not been measured well enough to warrant a thorough performance appraisal. Clustering + Java RMI + Web Services + Goodbye backward compatibility + Sweet IOR Indexing = YMMV (Your Mileage May Vary), there were just too many variables present in the test environment to sort through the various configurations and report meaningful results. If you've conducted a performance analysis and documented your findings feel free to send us your findings. In the meantime, we can make a few observations based on the limited testing we've conducted to date in the next sections.

8.1 What Performance Characteristics Can I Expect from the Latest Version?

The V1.2.5 release candidate we distributed a few months ago performed approximately 20% slower than version 1.2.0.1 for basic publish and subscribe. V1.2.6 Has narrowed that gap quite a bit, and when all results have been compiled, a single cluster configuration performs at worst the same as, but in most cases noticeably better than 1.2.5. Single node performance will still degrade a bit if you've set the Instrumentation services on and are reporting statistics at high rates, and degrade even more if you're running the Google-like search processing in the background. IOR Delete, Archive, and Query performance has improved immensely due to 3rd party software upgrades and our own numerous optimizations.

We are just scratching the surface in determining performance gains associated with using a multi-node clustered configuration. The value of this architecture is it can be optimally tuned based on expected usage of the core services. We will report on the results and even make several recommended configurations to the architecture for general usage categories.

8.2 Comparing the Latest to the Previous Version

See the discussion in the previous section.

8.3 Test Methodology

There are a number of basic tests AFRL performs in-house to evaluate RI versions and other JBI implementations from both a performance and functional perspective. Each of these test types are explained in a little more detail.

8.3.1 Performance Test Methodology

No formal performance tests were performed for this specific release as informal tests showed results from "the same as" -> "slightly better than" the performance of V1.2.5.

8.3.2 Functional Test Methodology

In order to provide a robust RI that is both useful and usable, much effort has been expended to enhance our functional testing methodologies and practices internally. Version 1.2.6 of the Core Services was tested in many ways:

- JUnit tests were developed to act as mock CAPI clients – both good and ill-behaved ones. The obvious intention being to ensure the core services function correctly under expected use and to ensure they handle unexpected usage gracefully.

- For cases when CAPI clients may not exercise a specific module, individual core components (the XPath to SQL parser for example) have their own set of exhaustive JUnit tests.

- Efforts are currently underway to automate (to the extent possible) JUnit testing with a build and test server jacked right into our configuration management build tree. When completed, this will essentially allow our developers to have daily feedback on the functional stability of the latest platform build.

- Not all portions of the core services can be faithfully tested via JUnit. These portions require more laborious manual testing at present, although our methods are documented and available to the curious.

Although much work has been done to ensure the RI is as robust as possible we realize the really determined bugs, some of which could be buried deep in the bowels of our own source, or that of one of the many third party libraries we use, will slip through the cracks. To that end should you come across any bugs, or have any feedback on aberrant cases and how the core services could better handle them, we encourage you to document the issue aggressively and submit a Software Problem Report or Change Request through CMDB.

Appendix A - Base Object Type

The base object metadata contains seven mandatory elements and two optional elements.

The JBI V1.2.6 RI implementation

RI V1.2.6 now supports XML Namespaces and has selected two namespaces for internal use. These two namespaces are **sys:** and **im:**.

The **sys** namespace is used internally by the Information Management Core Services to distinguish high level core services provided wrapper elements within the metadata. These high level elements are generated by the core services at publication time and include: **Metadata**, **Extended**, and **Published**. The **Metadata** element is the root element of the complete metadata XML document. The **Extended** element contains all metadata elements specifically generated by the information management core services. Currently this is limited to the BaseObject element, but future uses may include security tagging among other things. The **Published** element contains the client-provided metadata as-is.

The overall look of the platform wrapped metadata will be as follows:

```
<sys:Metadata>
  <sys:Extended>
    <im:BaseObject>
      PLATFORM PROVIDED BASEOBJECT INFORMATION STORED HERE!
    </im:BaseObject>
  </sys:Extended>
  <sys:Published>
    YOUR METADATA STORED HERE!
  </sys:Published>
</sys:Metadata>
```

The BaseObject is considered an element in the overall platform wrapped schema of the metadata. The xsd schema is as follows:

```
<xs:element xmlns:im=http://jbischemas.rl.af.mil/schemas/im name="im:BaseObject">
  <xs:annotation>
    <xs:documentation>
      Comment describing your root element
    </xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:all>
      <xs:element name="im:InfoObjectType">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="im:Name" type="xs:string"/>
            <xs:element name="im:MajorVersion" type="xs:positiveInteger"/>
            <xs:element name="im:MinorVersion" type="xs:nonNegativeInteger"/>
          </xs:sequence>
        </xs:complexType>
```

```
    </xs:element>
    <xs:element name="im:PublicationTime" type="xs:dateTime"/>
    <xs:element name="im:InfoObjectID" type="xs:string"/>
    <xs:element name="im:PublisherID" type="xs:string"/>
    <xs:element name="im:PlatformID" type="xs:string"/>
    <xs:element name="im:PayloadFormat" type="xs:string"
            minOccurs="0"/>
    <xs:element name="im:TemporalExtent" minOccurs="0">
     <xs:complexType>
      <xs:choice>
       <xs:element name="im:Instantaneous" type="xs:dateTime"/>
       <xs:sequence>
        <xs:element name="im:From" type="xs:dateTime"/>
        <xs:element name="im:To" type="xs:dateTime"/>
       </xs:sequence>
      </xs:choice>
     </xs:complexType>
    </xs:element>
    <xs:element name="im:Signature" minOccurs="0" type="xs:string"/>
   </xs:all>
  </xs:complexType>
</xs:element>
```

The following elements are the actual representations of the InfoObjects type and version that are set through the CAPI.

im:Name = represents the information object type name
im:MajorVersion = represents the information object type's major version number.

im:MinorVersion = = represents the information object type's minor version number.

The rest of the elements are set by the platform for internal use. Some may not be used but are placeholders for future implementations. Of special interest are the im:PublicationTime (the time the InfoObject was published)
Currently im:PayloadFormat is not used, however it is a place holder for future use. The client should not rely on this element being set with any information regarding payload format. If a client requires this functionality then it should create this or a similar element within their own schema (outside of the im namespace).

The im:TemporalExtent is also not in use but may be in the future. This element would be used, in part, to decide whether or not the system should continue to maintain the object within the repository. The decision to cull the space would be based on this parameter in concert with other established and enforced information management policies.

The following is an example of a platform provided metadata that has been returned by a JBI V1.2.6 platform.

```
 <im:BaseObject xmlns:im="http://jbischemas.rl.af.mil/schemas/im">
        <im:InfoObjectType>
                <im:Name>mil.af.rl.jbi.training.xmlxpath</im:Name>
```

<im:MajorVersion>1</im:MajorVersion>
<im:MinorVersion>0</im:MinorVersion>
</im:InfoObjectType>
<im:PublicationTime>2005-08-24T13:32:54Z</im:PublicationTime>
<im:InfoObjectID>4b6s1og-qvabzo-ecrp7frx-1-ecrv9bcc-6yl</im:InfoObjectID>
<im:PublisherID>4b6s1o1j-x6r1q4-ecrozxqx-1-ecrv79cm-6yl</im:PublisherID>
<im:PlatformID>afrl.j2ee</im:PlatformID>
</im:BaseObject>